НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ
ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
"___" _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системне програмування та спеціалізовані комп'ютерні системи»

спеціальності **123 «Комп'ютерна інженерія»**

на тему: «Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі щоденник» _____

Виконав:

студент IV курсу, групи КВ-12
(шифр групи)

Зимовець Єгор Олександрович _____
(прізвище, ім'я, по батькові) (підпис)

PhD. Тесленко О.К. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

(підпис)

Київ – 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ
ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних

систем Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
«___» червня 2025 р.

ЗАВДАННЯ
на дипломний проєкт студента
Зимовець Єгор Олександрович
(прізвище, ім’я, по батькові)

1. Тема проєкту Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі щоденник
__, керівник проєкту проф. каф. СПіСКС, д.т.н. проф Тесленко О.К. ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «25» травня 2025р. №1331-С

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту : див. ТЗ.

4. Зміст пояснювальної записки:

   1) аналіз існуючих рішень та обґрунтування теми дипломного проєкту;
   2) інструменти та методи розробки;
   3) аналіз та розробка програмного продукту;

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо):
   - Архітектура мобільного додатку.Структурна схема.
   - Алгоритм додавання нотаток. Схема алгоритму.
   - Інтерфейс головної сторінки/екрана створення нотаток. UI Макет.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н.		

7. Дата видачі завдання 30 жовтня 2024р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів	Примітка
1.	Видача завдання на дипломне проєктування	30.10.2024	
2.	Вивчення літератури за тематикою проєкту	17.11.2024	
3.	Аналіз існуючих рішень	20.02.2025	
4.	Розробка архітектури графічного інтерфейсу	15.03.2025	
5.	Тестування швидкості обробки даних	24.03.2025	
6.	Підготовка матеріалів графічної частини проєкту	10.04.2025	
7.	Оформлення документації дипломного проєкту	16.05.2025	

Студент _____ (підпис) Зимовець Є.О. (ініціали, прізвище)

Керівник проєкту _____ (підпис) Тесленко О.К. (ініціали, прізвище)

АНОТАЦІЯ

У дипломній роботі розглянуто процес розробки Адаптивного застосунку онлайн-організації щоденних завдань в хмарному середовищі для платформи Android з використанням сучасних технологій інтерфейсу Jetpack Compose та хмарного сервісу Firebase.
Основна мета полягає у створення зручного та інтуїтивно зрозумілого застосунку для ведення щоденних записів з можливістю зберігання данних у хмарі.

Розробка застосунку скаладалась з таких етапів:аналіз існуючих рішень , проєктування структури бази даних у Firestore, реалізація авторизації користувача за допомогою Firebase Authentication, створення інтерфейсів для додавання, редагування та перегляду записів, а також організація взаємодії між рівнями UI та логіки за архітектурною моделлю MVVM.

Застосунок дозволяє користувачеві створювати записи, зберігати їх у хмарі, сортувати за датою та керувати власними записами в зручному інтерфейсі. У процесі реалізації було застосовано мову Kotlin та середовище Android Studio.

Ключові слова: адаптивний застосунок, Firebase, Jetpack Compose, Android, Kotlin, MVVM, Firestore.

Abstract

The thesis examines the process of developing an Adaptive online application for organizing daily tasks in a cloud environment for the Android platform using modern technologies of the Jetpack Compose interface and the Firebase cloud service.
The main goal is to create a convenient and intuitive application for keeping daily records with the ability to store data in the cloud.

The application development consisted of the following stages: analysis of existing solutions, design of the database structure in Firestore, implementation of user authorization using Firebase Authentication, creation of interfaces for adding, editing, and viewing records, as well as organization of interaction between UI and logic levels using the MVVM architectural model.

The application allows the user to create records, store them in the cloud, sort by date, and manage their own records in a convenient interface. The implementation process used the Kotlin language and the Android Studio environment.

Keywords: responsive application, Firebase, Jetpack Compose, Android, Kotlin, MVVM, Firestore.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.002 ТЗ	Адаптивний застосунок	4		
			Онлайн організації			
			Щоденних задач у			

			Хмарному середовищі			
			Технічне завдання			
	А4	ІАЛЦ.045440.003 ТП	Адаптивний застосунок	1		
			Онлайн організації			
			Щоденних задач у			
			Хмарному середовищі			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045440.004 ПЗ	Адаптивний застосунок	50		
			Онлайн організації			
			Щоденних задач у			
			Хмарному середовищі			
			Пояснювальна записка			
	А4	ІАЛЦ. 045440.005 Д1	Структурна схема	1		
			класів програми.			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ. 045440.001 ОА		
Розробив		Зимовець Є.О.			Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі **Відомість технічного проекту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Тесленко О.К.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-12	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ. 045440.006 Д2	Алгоритм реєстрації	1		
			Та авторизації			
			Користувача			
			Схема алгоритму			
	А4	ІАЛЦ. 045440.007 Д3	Алгоритм фільтрації	1		
			задач			
			Схема алгоритму			
	А4	ІАЛЦ. 045440.008 Д4	Алгоритм нагадування	1		
			Про невиконані задачіі			
			Схема алгоритму			

					ІАЛЦ.045440.001 ОА	Арк
Змін.	Арк.	№ докум.	Підпис	Дата		2

ЗМІСТ

					ІАЛЦ. 045440.002 ТЗ		
Зм	***Лист***	***№ докум.***	***Підп.***	***Дата***			
Розроб.		Зимовець Є.О.			*Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі* ***Технічне завдання***	***Літ.***	***Лист*** / ***Листів***
Перев.		Тесленко О.К.					1 / 4
Н. контр.		Клятченко Я. М				КПІ ім. Ігоря Сікорського, ФПМ, КВ-12	
Затв.		Романкевич. В.О					

НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі».

Галузь застосування: мобільні інформаційні системи.

ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є розробка Адаптивного застосунок онлайн-організації щоденних завдань в хмарному середовищі для створеня зберігання та сортування особистих записів.

ДЖЕРЕЛА РОЗРОБКИ

Офіційна документація Android (Jetpack Compose, Firebase),статті та приклади з відкритих джерел (Medium, Stack Overflow, GitHub), науково-технічна література з архітектури програмного забезпечення з відкритих джерел.

ТЕХНІЧНІ ВИМОГИ

Вимоги до програмного продукту, що розробляється

- Сумісність із мобільними пристроями (смартфони, планшети)
- Авторизація через Firebase Authentication;

Вимоги до апаратного забезпечення

- Процесор: 4-ядерний, не нижче Intel Core i5 або аналог AMD Ryzen 5;
- Оперативна пам’ять: 8 Гб;
- Дисплей з роздільною здатністю не менше 1280x720

Вимоги до програмного та апаратного забезпечення користувача

- Мобільний пристрій: смартфон або планшет з мінімум 2 ГБ оперативної пам’яті та 200 МБ вільного місця на пристрої;
- Операційна система: Android 8.0 (Oreo) або новіша;
- Платформи: Android 8.0 (API 26) і вище;

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	ІАЛЦ.045440.000	Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ. 045440.001 ОА	Опис альбому	2	
3	А4	ІАЛЦ. 045440.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ. 045440.003 ТП	Відомість технічного проєкту	1	
5	А4	ІАЛЦ. 045440.004 ПЗ	Пояснювальна записка	51	
6	А4	ІАЛЦ. 045440.005 Д1	Структурна схема класів програми.	1	
7	А4	ІАЛЦ. 045440.006 Д2	Алгоритм реєстрації та авторизації користувача Схема алгоритму	1	
8	А4	ІАЛЦ. 045440.007 Д3	Алгоритм фільтрації задас Схема алгоритму	1	
9	А4	ІАЛЦ. 045440.008 Д4	Алгоритм нагадування про невиконані задачі Схема алгоритму	1	

					ІАЛЦ.045440.003 ТП
Змін.	Арк.	№ докум.	Підпис	Дата	
Розробив		Зимовець Є.О.			*Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі* ***Відомість технічного проєкту***
Перевірив		Тесленко О.К.			
Консульт.					
Н. контроль		Клятченко Я.М.			
Зав. каф.		Романкевич В.О.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського, ФПМ КВ-12

Пояснювальна записка
до дипломного проєкту

на тему: “Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі”

Київ – 2025

ЗМІСТ

					ІАЛЦ. 045440.04 ПЗ			
Зм	***Лист***	***№ докум.***	***Підп.***	***Дата***				
Розроб.		Зимовець Є.О.			Адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі ***Технічне завдання***	***Літ.***	***Лист***	***Листів***
Перев.		Тесленко О.К.					1	4
Н. контр.		Клятченко Я. М				КПІ ім. Ігоря Сікорського, ФПМ, КВ-12		
Затв.		Романкевич. В.О						

Перелік скорочень, умовних позначень, термінів

UI (User Interface) – Користувацький інтерфейс

UX (User Experience) – Досвід користувача

Jetpack Compose – Сучасний інструмент побудови інтерфейсів користувача для Android

Firebase – Хмарна платформа Google для зберігання даних, авторизації, аналітики тощо

Firestore – Хмарна база даних реального часу від Firebase

Firebase Authentication – Сервіс для реєстрації та входу користувачів

Material Design 3 – Стиль оформлення інтерфейсів від Google

MVVM (Model-View-ViewModel) – Архітектурний шаблон для розділення логіки додатку

Вступ

У сучасному світі цифрові технології дедалі глибше інтегруються в повсякденне життя,змінюючи та покращуючи методи навчання,роботи та організації інформації. Одним із важливих напрямів цифровізації є автоматизація процесів самоменеджменту, планування та контролю особистих справ.Враховуючи великий рівень навантаженості людини в сучасному світі ми потребуємо зручний інструмент для створення,пошуку та організації записів,нотаток.

Застосунки для організації записів надають людині можливість контролювати та відстежувати своє життя.Оскільки це переплітається з усіма сферами життя людини,адже людина може зберігати інформацію про все застосунок повинен бути надійним,швидким та ефективним.Через швидкий розвиток технологій вимоги стають вище з кожним днем і тому розробка нового адапативного додатку для організації справ є актуальною.

У рамках дипломної роботи було розроблено адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі для ОС Android, що реалізує функціональність онлайн щоденника з використанням сучасного інструментарію Jetpack Compose та хмарного середовища Firebase. Програма дає користувачу можливісь зберігати,сортувати та видаляти записи,а також синхронізувати данні у хмарі.

Метою даної дипломної роботи є створення адаптивного застосунок онлайн-організації щоденних завдань в хмарному середовищі, який би відповідав сучасним стандартам, продуктивності, безпеки зберігання даних і розширюваності. У процесі розробки було досліджено архітектурні підходи (MVVM), реалізовано основні екрани додатку, забезпечено інтеграцію з Firebase Authentication та Firestore, а також

а також протестовано роботу системи на реальних пристроях.

Актуальність теми зумовлена постійним зростанням кількості мобільних користувачів і потребою у персональних цифрових інструментах для самоорганізації. Результати цієї роботи можуть бути використані як основа для розробки більш складних інформаційних систем, сервісів персонального моніторингу, а також як навчальний приклад інтеграції сучасних технологій Android-розробки.

3.

1.1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЄКТУ

1.1 Постановка задачі

У сучасному цифровому суспільстві питання ефективної організації особистої інформації, ведення щоденників, планування завдань і збереження нотаток виходить на перший план. Швидкий темп життя, збільшення обсягу інформації, яку щодня доводиться обробляти людині, потребує нових інструментів для її структурування, контролю та швидкого доступу. Традиційні паперові щоденники втрачають актуальність, поступаючись місцем мобільним додаткам, які забезпечують зручність, мобільність та функціональність.

Одним з ефективних підходів до розв’язання цієї задачі є розробка адаптивного застосунку онлайн-організації щоденних завдань в хмарному середовищі, що дозволяє користувачам створювати, редагувати, переглядати та видаляти особисті записи, зберігаючи їх у хмарному середовищі. Такий додаток має на меті не лише організацію особистих записів а ще й їх захист та швидкий доступ до них в будь який момент часу та в будь-якшму місці, а також підтримку інтуїтивно зрозумілого інтерфейсу користувача.

Сучасні засоби Android-розробки надають потужний інструментарій для реалізації подібних проєктів. Зокрема, Jetpack Compose дозволяє створювати адаптивні, красиві та продуктивні інтерфейси користувача з меншими витратами часу, а хмарна платформа Firebase — реалізовувати функції автентифікації користувачів, зберігання даних у реальному часі

Firebase виступає у ролі бекенд-сервісу, що звільняє розробника від необхідності створення окремого серверного середовища. Це суттєво спрощує реалізацію бази даних у реальному часі (Cloud Firestore), а також дозволяє обробляти авторизацію, надсилати push-сповіщення й масштабувати додаток при зростанні кількості користувачів.

Важливою особливістю системи є її кросплатформеність у контексті хмарного зберігання даних, що забезпечує доступ до записів користувача незалежно від пристрою, на якому встановлено додаток. Для цього реалізується синхронізація даних з Firebase Firestore у режимі реального часу. Окрім основного функціоналу, розробка також враховує питання продуктивності, мінімального споживання ресурсів пристрою, а також адаптацію під різні розміри екранів.

В умовах зростаючого попиту на персоналізовані рішення для самоорганізації, подібні застосунки мають велике прикладне значення. Їх можна використовувати не лише для ведення особистих нотаток, а й у освітній сфері (щоденники учнів, записи розкладу занять), професійній діяльності (робочі плани, задачі), а також у медичних і психологічних практиках (щоденники настрою, харчування).

Усе це зумовлює актуальність теми дипломного проєкту, а також його практичну цінність як приклад інтеграції сучасних Android-технологій та хмарних сервісів у корисний мобільний продукт. Результати цієї роботи можуть бути використані як основа для створення комерційного застосунку, подальшої розробки функціоналу (наприклад, додавання голосового введення, нагадувань, експорту в PDF тощо), а також у навчальних цілях для демонстрації взаємодії між клієнтською частиною й хмарним бекендом.

1.2.Аналіз існуючих засобів

У сучасному цифровому середовищі існує велика кількість програмних рішень, що виконують функцію адаптивного застосунку онлайн-організації щоденних завдань в хмарному середовищі. Більшість із них представлені у вигляді мобільних застосунків, доступних на платформах Android та iOS. Ці застосунки дозволяють користувачам фіксувати особисті нотатки, планувати завдання, зберігати ідеї, вести емоційний або харчовий щоденник, відслідковувати прогрес та багато іншого. Незважаючи на те, що на ринку вже присутні численні продукти, кожне з рішень має як переваги, так і певні обмеження, які створюють передумови для розробки нових, більш адаптивних, зручних і персоналізованих додатків.

2.1. Популярні мобільні застосунки для ведення щоденника

1. Journey

Journey — це один із найпопулярніших мобільних щоденників, який дозволяє зберігати текстові записи, додавати фото, відео, мітки геолокації. Додаток підтримує хмарну синхронізацію через Google Drive, має версії для Android, iOS та веб-інтерфейс. Він також дозволяє захищати записи за допомогою пароля або біометрії.

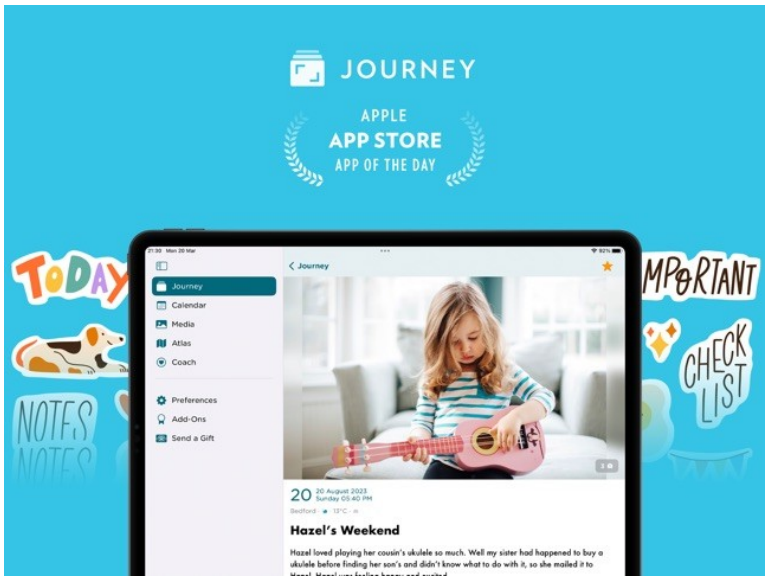

Рисунок 1.1 Journey app

Переваги:

- Багатофункціональність;
- Синхронізація між платформами;
- Інтерфейс користувача високої якості.

Недоліки:

- Значна частина функцій доступна лише у платній версії;
- Важкий інтерфейс для користувачів, яким потрібна лише базова функціональність.

2. Day One

Day One — преміум-застосунок для ведення щоденника, який відзначено нагородами. Він забезпечує надзвичайно привабливий інтерфейс, підтримує додавання фотографій, аудіозаписів, автоматичне визначення погоди, місця та ін

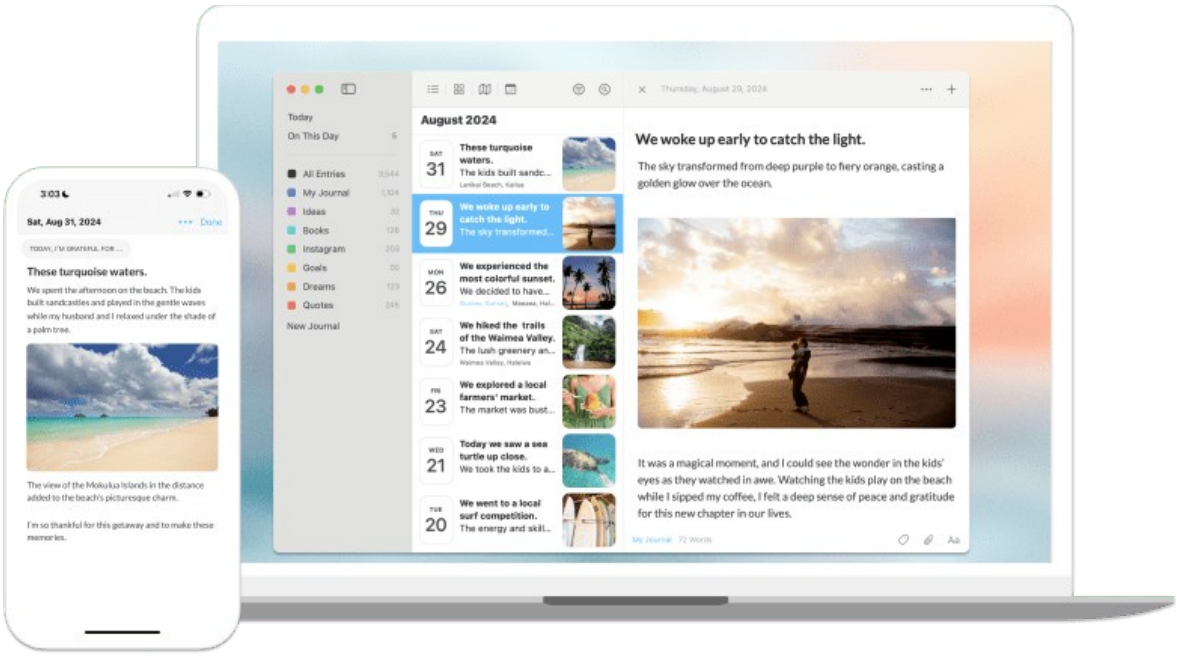

Рисунок 1.2 Day one app

Переваги:

- Високий рівень інтеграції з екосистемою Apple;
- Хмарне резервне копіювання;
- Надійне шифрування даних.

Недоліки:

- Працює переважно в екосистемі iOS/macOS;
- Повна функціональність потребує передплати;

- Обмежена кастомізація інтерфейсу під особисті потреби.

3. Diaro

Diaro — ще один застосунок, орієнтований на створення щоденникових записів із можливістю класифікації за категоріями, тегами, датами. Підтримує синхронізацію з Dropbox.

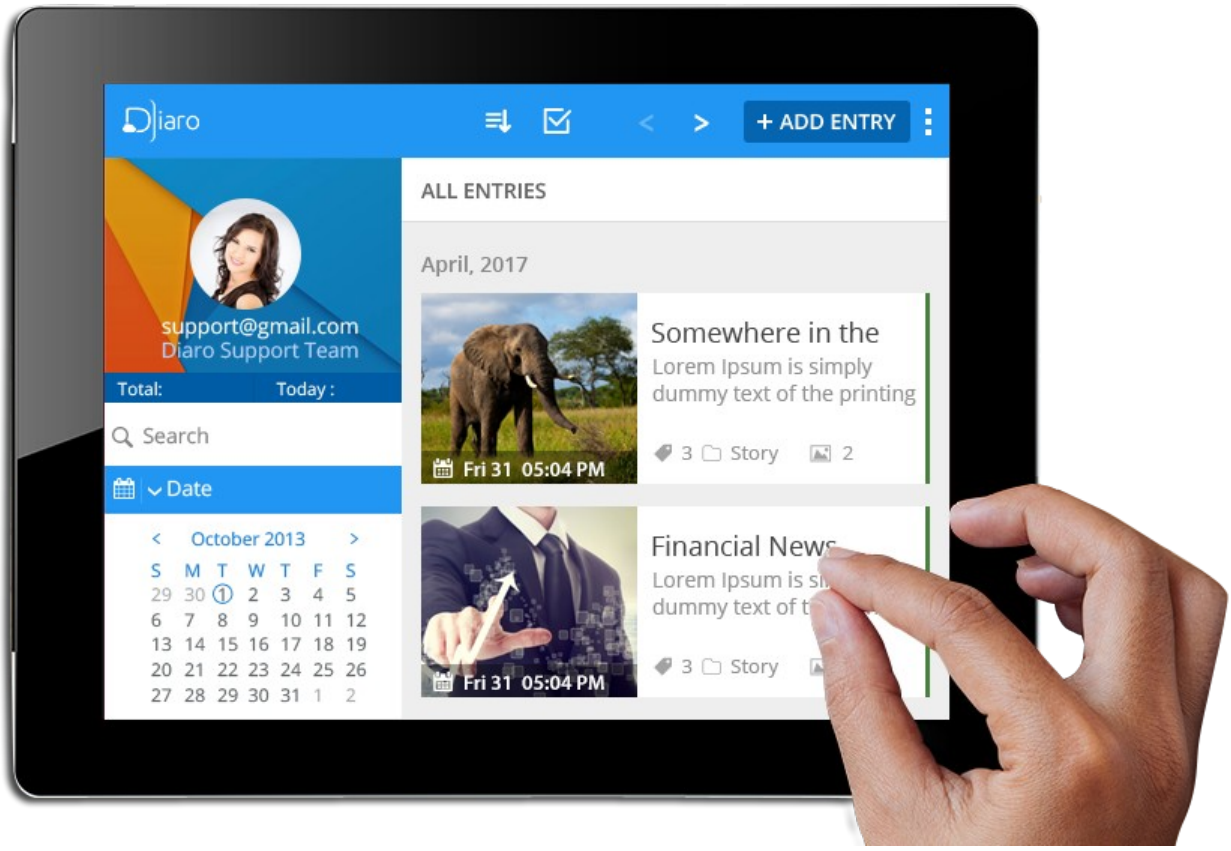

Рисунок 1.3 Diaro app

Переваги:

- Простота у використанні;
- Функція експорту в PDF;
- Пошук за ключовими словами.

Недоліки:

- Обмежений візуальний стиль;
- Відсутність активної підтримки нових технологій (наприклад, голосового введення, інтеграції з календарем).

4.Google Keep / Evernote

Хоча ці застосунки не є щоденниками у прямому сенсі, багато користувачів адаптують їх під особисті записи. Google Keep дозволяє створювати нотатки, чек-листи, нагадування; Evernote — більш потужний інструмент для організації знань.

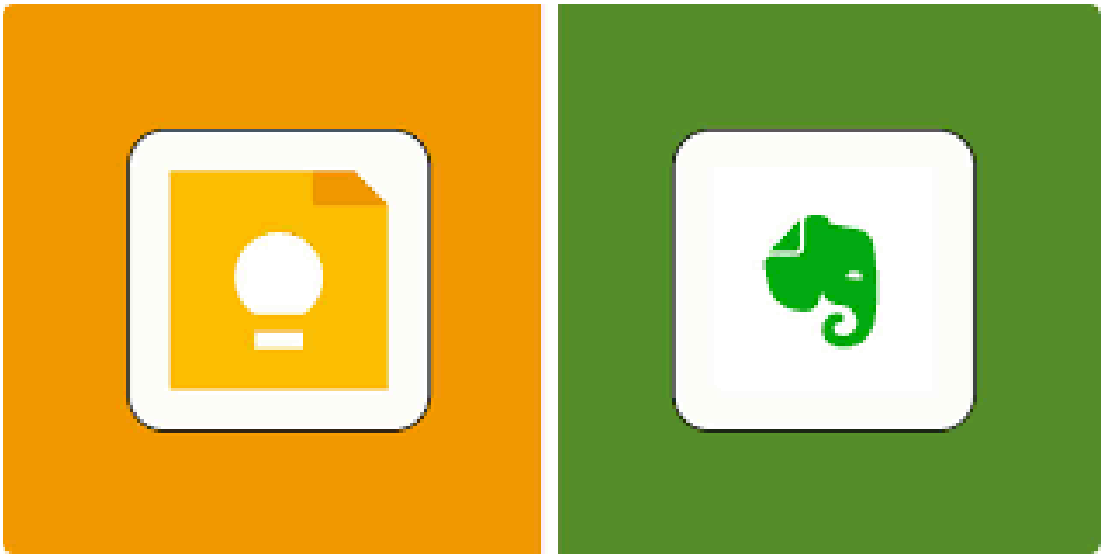

Рисунок 1.4 Google Keep / Evernote

Переваги:

- Інтеграція з іншими сервісами Google;
- Кросплатформеність;
- Підтримка голосових нотаток та зображень.

Недоліки:

- Відсутність логіки «щоденника» як послідовності подій/днів;
- Менший акцент на приватність та персоналізацію.

1.3 Обґрунтування теми дипломного проєкту

У сучасному світі мобільні технології стали невід'ємною частиною повсякденного життя. Смартфони та планшети активно використовуються не лише для комунікації, а й для ведення особистих нотаток, планування завдань, контролю за настроєм, станом здоров'я та іншими аспектами життя. У зв'язку з цим значно зріс попит на універсальні, зручні, безпечні

та персоналізовані мобільні щоденники. Однак попри велику кількість наявних програмних рішень, більшість із них мають низку недоліків, що стримують їх повноцінне використання широким колом користувачів, зокрема в українському інформаційному просторі.

Найбільш поширеним обмеженням сучасних цифрових щоденників є їхня залежність від хмарних сховищ і постійного підключення до інтернету. У багатьох випадках дані користувача автоматично синхронізуються із зовнішніми серверами, що породжує ризики втрати приватності, порушення конфіденційності та витоку персональної інформації. У контексті зростаючої уваги до цифрової безпеки, такі моделі обробки даних є недостатньо надійними, особливо якщо мова йде про особисті щоденникові записи, які можуть містити емоційно чутливу або навіть інтимну інформацію.
Окрему проблему становить відсутність локалізованих рішень українською мовою. Більшість популярних щоденників орієнтовані на глобальний англомовний ринок, ігноруючи особливості місцевих користувачів. Це створює додаткові бар’єри для людей, які прагнуть комфортного, рідномовного інтерфейсу, або ж просто надають перевагу українському цифровому продукту.
Ще одним вагомим аргументом на користь створення нового програмного рішення є відсутність повнофункціональних безкоштовних інструментів, які не обмежували б користувача у використанні ключових можливостей (наприклад, синхронізації, захисту даних, створення категорій і тегів тощо). У багатьох випадках користувач змушений купувати підписку, щоби отримати доступ до базових функцій, що суперечить концепції відкритого і масового використання додатка, особливо в умовах економічних обмежень.

Зважаючи на ці виклики, було прийнято рішення адаптивний застосунок онлайн-організації щоденних завдань в хмарному середовищі який буде:

- працювати локально з можливістю офлайн-збереження нотаток;
  підтримувати синхронізацію через Firebase, але без жорсткої залежності від неї;
- мати українську локалізацію та інтуїтивний інтерфейс;
- використовувати сучасні інструменти, такі як Jetpack Compose, Material 3, Firebase Authentication та Firestore;
- забезпечувати приватність користувача без необхідності передавати дані на сторонні сервери без згоди;
- бути безкоштовним, з відкритим кодом і можливістю розширення функціоналу.

Важливо також зазначити, що психологічний ефект від ведення щоденника є науково підтвердженим: регулярні записи допомагають зменшити рівень стресу, покращити самоаналіз, організувати думки та планувати особистий розвиток. Це створює не лише технічне, а й соціальне підґрунтя для актуальності такого застосунку.

Крім того, запропонований проєкт спрямований на формування навичок розробки повноцінних мобільних застосунків, що охоплюють як клієнтську, так і серверну частину. В умовах популярності архітектур типу "serverless", застосування Firebase як BaaS (Backend-as-a-Service) дозволяє значно прискорити розробку та сконцентруватися на якості інтерфейсу та функціональності. Це не лише актуалізує тему з точки зору користувача, а й відповідає трендам сучасної мобільної інженерії.

Таким чином, обрана тема дипломного проєкту є актуальною, та корисною. Вона дозволяє задовольнити усі потреби користувачів включаючи ьезпечність та надійність.Результатом стане адаптивна платформа для організації щоденних завдань яка у перспективі може розширитись і зайняти велику частину цієї сфери.

Окрім технічної складової, проєкт має і прикладне значення, оскільки сприяє покращенню особистої продуктивності та тайм-менеджменту користувачів. Створення гнучкого інтерфейсу із синхронізацією даних у реальному часі дозволяє забезпечити безперервний доступ до задач, незалежно від пристрою чи локації. Такий підхід особливо цінний в умовах сучасного динамічного стилю життя, де ефективна самоорганізація відіграє ключову роль. У перспективі платформа може бути інтегрована з календарями, голосовими асистентами та іншими сервісами, що підвищить її конкурентоспроможність і практичну цінність.

1. 2. ТЕХНОЛОГІЧНІ ТА ПРОГРАМНІ ЗАСОБИ ДЛЯ РОЗРОБЛЕННЯ ЗАСТОСУНКУ

2.1 Класифікація мобільних застосунків

У сучасному світі з цифровізацією в усіх сферах життя мобільні додатки для управління особистими цілями та щоденними справами є важливими для самоорганізації, ефективного управління часом та планування. Ці програми зазвичай використовуються у сферах навчання, роботи, побутового життя та психічного здоров'я. Вони можуть бути відсортовані за різними характеристиками: за тим, як вони налаштовані, виходячи з їх роботи, тим, що вони роблять, тим, як вони зберігають інформацію, а також наскільки вони добре відповідають потребам користувача.

За методом встановлення

Нативні програми

Це програми, зроблені для конкретної ОС (наприклад, Android або iOS) наприклад за допомогою таких інструментів як Kotlin/Java для Android, Swift/Objective-C для iOS. Вони можуть використовувати всі функції пристрою - відстеження розташування, знімок фотографій, сканування відбитків пальців та додаткові функції. Такі програми забезпечують найкращу продуктивність, стабільність та оптимізацію інтерфейсу

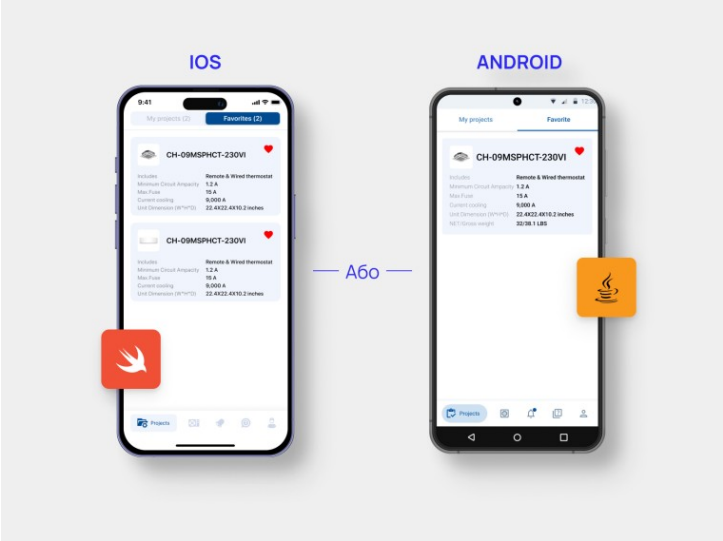

Рисунок2.1 Приклад Нативних додатків

Кросплатформені додатки

Використовуйте спільний код для різних систем (наприклад, Flutter, React Native, Xamarin). Створення коштує менше, а вдосконалення швидше. Тим не менш, певні обмеження можуть застосовуватися до операцій, що стосуються операцій які залежать від платформи.Такий підхід часто використовують у стартапах та MVP застосунках

Веб-застосунки (або прогресивні веб-додатки-PWA)Вони працюють. через веб -браузер і не потребують встановлення програми. – імітують деякі функціонал нативного додатка, та можуть функціонувати офлайн.

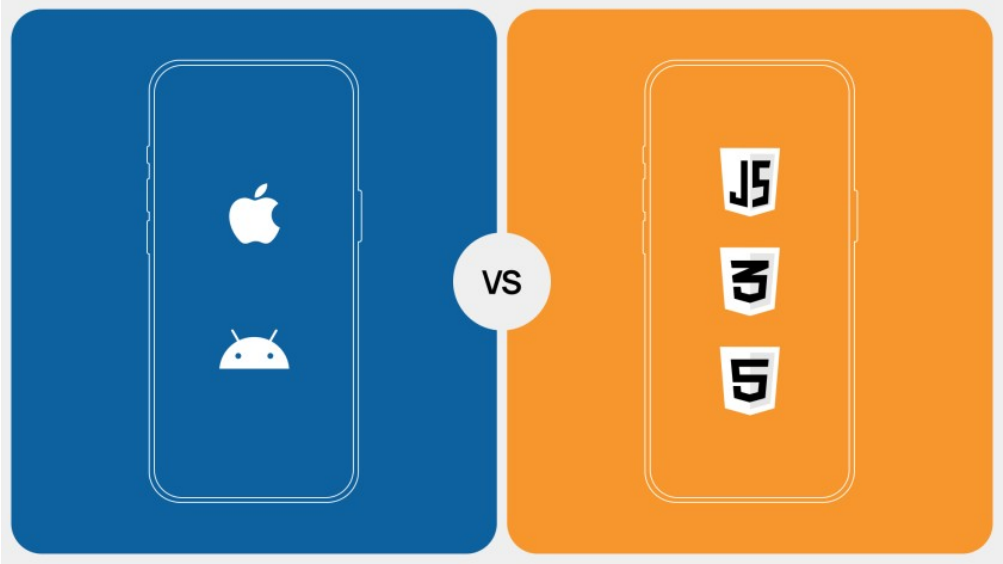

Рисунок 2.2 Різниця нативних і кросплатформених застосунків

За способом збереження та обробки даних

- Локальні застосунки

Усі дані зберігаються виключно на пристрої людини. Вони можуть повністю функціонувати без Інтернету, але якщо пристрій буде неправильно розміщений або пошкоджений, інформація може бути втрачена. Недоліком є те, що він вимагає багато зусиль.

- Хмароорієнтовані застосунки

Основні дані зберігаються на комп'ютері або у віддаленому просторі даних (наприклад, Firebase, AWS, iCloud). Переконайтесь, що пристрої синхронізуються, зберігають дані, дозволяють використовувати групу. Існує залежність від Інтернету.

- Гібридні застосунки

  Інформація зберігається на пристрої, а потім автоматично узгоджується з сервером при Інтернеті.Це ідеальний метод для гнучких та послідовних програм.

За функціональністю

- Прості нотатки та щоденники

  Надають лише можливість зберігати записи без будь якої організації

- Планувальники задач

  Вони пропонують допомогу для завдань, сповіщень, термінових тегів, годинників, повторних дій.Приклади: завдання Google, Todeist.

- Комплексні системи продуктивності

  Ви не тільки економите час на організації задач,а ще й отримуєте діаграми та поради.Часто поєднується з графіками, електронними листами та додатковими функціями.

2.

За рівнем адаптивності

- Статичні інтерфейси

  Інтерфейс однаковий для всіх користувачів, незалежно від пристрою чи умов використання.

- Адаптивні інтерфейси

  Макет змінюється відповідно до розміру екрана, обраної теми (темно/світла) та мови.Наприклад, шрифт змінює розмір щоб бути зручнішим для користувачів, або компоненти змінюють послідовність на планшетах.

- Персоналізовані інтерфейси

  Програма розглядає, як користувач діє та модифікує макет або надає функції на основі його дій (наприклад, AI пропонує повторити задачу, з якою у користувача часто виникають труднощі).

За сферою використання

- Особисті щоденники

  Журнали індивідуальних міркувань, споглядання, емоційних станів. Часто з шифруванням, біометрією.

- Платформи для організації навчання

  Їх використовують вчителі та учні для організації курсів, повних завдань.

  • Бізнес -додатки

  Організація проекту, обов'язки працівників, об'єднання системи CRM, JIRA

- Медичні або психологічні трекери
- Емоційні журнали які дозволяють відстежувати морадьний стан людини, відстежуюють її харчову поведінку або для детального опису розпорядку дня, часто використовуються під наглядом терапевту.

2.2 Архітектура «клієнт – хмара»

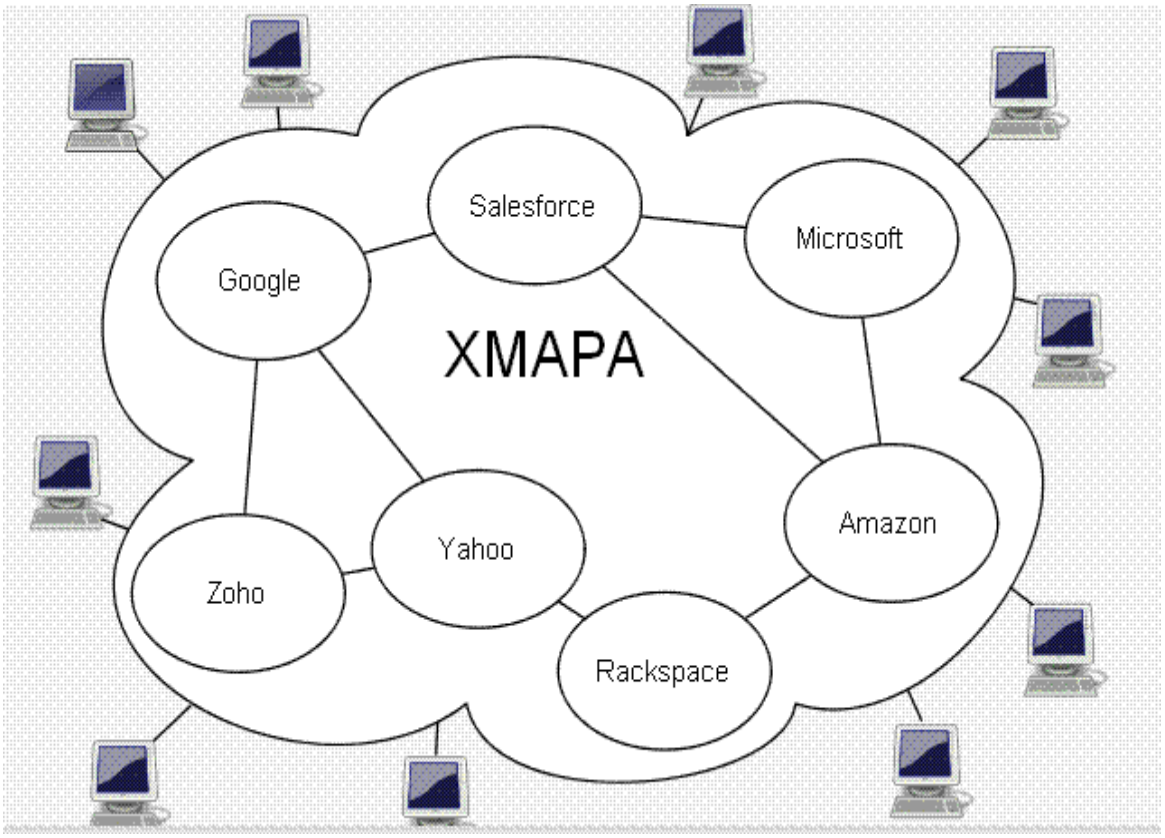

Рисунок 2.1 Візуалізація клієнт-хмара

Нові мобільні додатки стають більш складними та заплутаними, пропонуючи функції, які виходять за рамки просто введення або створення нотаток. Сьогоднішні користувачі передбачають програмне забезпечення, яке пропонує не просто фундаментальну ефективність, але й простоту використання, швидкість, самодостатність, безпеку та можливість зв'язуватися між гаджетами. Для досягнення цих цілей у розробці програмного забезпечення все більше впроваджуються хмарні технології, які реалізуються за допомогою структкри "клієнт – хмара".

Цей підхід працає за принципом розподілення на кліента який робить якісь дії у додатку та хмару,яка оброблює інформацію та забезпечує швидкість та гнучкість. Ця система має вирішальне значення для використання в особистих записах, завданнях, журналах, адже це забезпечує швидкий доступ до записів та головне захищає приватну інформацію.

У контексті платформ для управління щоденними діями в Інтернеті модель клієнта-сервер пропонує кілька важливих переваг. Спочатку він дозволяє користувачеві переглядати свої завдання та нотатки з будь-якого пристрою-смартфона, планшета, ноутбука, без прив'язки до певної платформою. По друге ця модель працює таким чином що будь які внесені зміни одразу оновлюютьсяі відображаються у всіх користувачів. По третє, завдяки тому що обробка данних відбувається на у хмарі, а не безпосередньо на пристрої то швидкість обробки швидше і є можливість одночасно обробляти запити від багатьох користувачів без втрати швидкості.Крім того, централізоване зберігання інформації в хмарному середовищі забезпечує високу надійність і захист даних, оскільки регулярно виконуються резервні копії та застосовуються механізми авторизації доступу. Завдяки цьому користувачі не ризикують втратити важливу інформацію навіть у разі втрати пристрою або пошкодження локального сховища. Модель клієнт-сервер також спрощує оновлення програмного забезпечення.

Головна ідея архітектури “клієнт – хмара”

Основна ідея архітектури клієнта-хмара полягає у обміні завданнями між інтерфейсом користувача (на пристрої користувача) та хмарними системами зберігання та обчислень. Простіше кажучи, всі складні обчислення, обробка даних та зберігання, а також їх пошук, не на гаджеті користувача, а у віддалених дата-центрах, доступні через Інтернет.

У цьому підход головне завдання мобільного додатку:

- Збір вхідних даних (від користувача)
- Відображення візуальної інформації (інтерфейс)
- Обмін даними на сервер через API (інтерфейси програмування).

Хмара — це складна система, що виконує роль бекенду (backend) застосунку, і саме вона:

- Керує ідентифікацією користувача (за допомогою автентифікації Firebase)
- Пропонує постійну експлуатацію послуг, незалежно від того, скільки людей користуються ними.
- Автоматично масштабується у міру збільшення навантаження,Дозволяє отримувати інформацію з будь -якого місця в усьому світі, з будь -якого гаджета.

Компоненти архітектури “клієнт – хмара”

Клієнт-хмарна архітектура передбачає три основні компоненти: клієнт, хмарний сервіс та мережу. Клієнт - це система (комп'ютер, мобільний пристрій), з якої користувач взаємодіє з хмарним сервісом. Хмарний сервіс - це програмне забезпечення та інфраструктура, які надаються хмарним провайдером (наприклад, AWS, Azure, Google Cloud). Мережа - це засоби (інтернет), що дозволяють клієнту та

хмарному сервісу взаємодіяти.

Докладніше про кожен компонент:

- Клієнт

  Це система (комп'ютер, мобільний пристрій), з якої користувач взаємодіє з хмарним сервісом.

  Використовує клієнтські програми (програми на мобільному пристрої, браузерні розширення) для взаємодії з хмарним сервісом.

- Хмарний сервіс

  Це програмне забезпечення та інфраструктура, які надаються хмарним провайдером.

  Надає користувачам доступ до різноманітних послуг (обчислювальні потужності, сховища даних, програмне забезпечення як послуга, тощо).

- Мережа

  Забезпечує зв'язок між клієнтом та хмарним сервісом.

  Зазвичай використовується інтернет для передачі даних між клієнтом та серверами хмарного провайдера.

Принципи взаємодії між клієнтом і хмарною інфраструктурою

Взаємодія між мобільним клієнтським застосунком і хмарною інфраструктурою в сучасних системах організації завдань відбувається завдяки мережевим протоколам, таким як HTTP(S) або WebSocket, а обмін даними здійснюється через Firebase SDK. У цій моделі клієнт ініціює запити до хмари, а сервер обробляє їх, виконує необхідні дії (читання, запис, оновлення, видалення) і повертає результат.

У системі онлайн-організації щоденних завдань основними аспектами цієї взаємодії є:

- Формування запиту

  Кожна дія користувача викликає певну частину коду яка відповідає за цю дію.

  Після цього формується JSON файл що описує структуру змін і за допомогою API виaddкликає потрібну функцію.Наприклад add(), update() у Firebase SDK
- Передача запиту до хмари

  Після того як запит був сформований він передається зашифрованим канлом через HTTPS чи Websocket до хмари.За допомогою Firebase додаток автоматично проводить автентифікацію користувача.
- Обробка запиту в хмарі

  При потраплнні у хмару данні перевіряються за протоколом (Firebase Security Rules). У разі проходження перевірки виконуються операції і усі нові данні заносяться в базу данних.
- Формування та повернення відповіді

  Після обробки запиту сервер формує відповідь. Коли відповідь сформована він надсилає її користувачу.Ця відповідь обов'язково містить інформацію про те чи вдалось обробити запити і якщо вдалось то також містить актуальну інформацію яку бажав отримати користувач.
- Оновлення інтерфейсу користувача

  Це фінальний етап результам якого є оновлення інтерфейсу вкористувача відповіно до запитів які він надіслав.Результатом може бути поява на екрані нової нотатки,видалення нотатки,перехід на іншу сторінку або будь які інші візуальні зміни.

Види архітектури в контексті мобільних хмарних застосунків

- Дворівнева архітектура

  Суть дворівневої архітектури полягає в тому що існує два рівні

клієнт і хмара. Клієнт посилає запит на пряму до хмари.

До переваг такої реалізації належать простота створення програми,неаелику кількість компонентів та мале навантаження на сервер.

До недоліків належить проблема з безпекою,адже прямий зв’язок з сервером не дуже надійний в плані безпеки.Ще одним недоліком є те що ця система спільна для всіх користувачів і якщо будуть неполадки то вона буде недоступною для кожного користувача.

- Трирівнева архітектура

Цей тип архітектури складається з трьох рівнів: клієнт,сервер додатку,сервер баз данних.Працює за принципом де кліент надсилає запит який оброблюється на сервері додатку,а сервер баз данних тільки оновлює та надсилає данні нахзад.

Через більшу кількість рівнів створюється більш чітка ієрархіє де кожен рівень відповідає за свою спеціалізовану задачу.

До переваг належить зменшення навантаження на сервер баз данних що покращує роботу застосунку також робить застосунок простішим для маштабування.Також робить навантаження на систему рівномірнішим що покращує ефертивність.

До недоліків належить складніша реалізація ніж у дворівневої архітектури та потреба у більшій кількості ресурсів.

2.3 Jetpack Compose як інструмент UI-розробки

Під час розробки будь якого мобільного застосунку дуже важливою часнитою є UI(user interface),через те що вся взаємодія користувача з сервером відбувається через нього.Від UI залежить перше враження користувача і це сильно впливає на те як активно він буде користуватись застосунком і чи взагалі буде.По цим причинам дуже важливо зробити його максимально зручним,гнучким та інтуїтивно

зрозумілим.

До недавнього часу для створення додатків використовувалась XML розмітка.Цей підхід полягає у ьому що структура усіх об’єктів вручну описувалачь в XML файлі а потім окрема описувалась їх логіка за допомогою таких мов програмування як Java/Kotlin.На жаль у цього підходу було багато недоліків. Основними з них є дуже велика кількість дубльованого коду.Також через такий підхід було складно підтримувати та розширювати додатки через велику кількість коду який треба було змінювати вручну.

На заміну такому підходу прийшов Jetpac Compose. Jetpac Compose це фреймворк який дозволяє одразу оптсувати вигляд і роботу компонентів мовою Kotlin. Це сильно покращує читабельність коду,що є дуже важливим адже пришвидшує та полегшує створення програм.Також це зменшує дубльованість коду і полегшує підтримку та розширення застосунків.

Архітектурні особливості Jetpack Compose

Jetpac Compose побудований по принципу композиції.Це означає що кожна Composable-функція є окремою частиною інтерфейсу.Це дозволяє легко відстежувати помилки і швидко тестувати програму.Також в Jetpac Compose не обов’язково запускати всю програму щоб перевірити якусь окрему Composable-функцію,можна одразу запустити саме її і тільки її.

Для зміни інтерфейсу не потрібно викликати функцію notifyDatasetChanged() або подібні.Адже в Jetpac Compose інтерфейс змінюється в той момент як відбувається зміна стану.

Основні інструменти Compose:

- @Composable — анотація, яка позначає функцію як UI-компонент,тобто функція без цієї анотації не матиме можливості відображатись на екрані.

- State / MutableState — типи змінних, які автоматично оновлюють інтерфейс при зміні значення.
- remember — оператор для збереження стану всередині Composable-функції.
- LaunchedEffect, DisposableEffect — механізми для керування побічними ефектами й життєвим циклом.
- Scaffold, Column, Row, Box — базові компоненти для компонування структури екрана.
- Navigation — система керування маршрутами між екранами.

Основні переваги Jetpack Compose

- Покращення функціональності і зменшення коду

  Через відмову від XML сильно зменшилась кількість коду,крім того тепер через відсутність адапторів та шаблонів які використовувались щоб пов’язати XML з Kotlin кодом весь код який відповідає за свою частину інтерфейсу став більш зрозумілим і читабельним що робить написання коду легшим,швидшим і більш ефективним.

- Адаптивність до різних пристроїв

  Добре підлаштовується під різні параметри пристроїв,наприклад розміри екрану та під деякі системні налаштування.

- Прев’ю та тестування

  Це ідеальний інструмент для тестування адже за допомогою анотації @Preview не обов’язково запускати всю програму на емуляторі,можна запустити окремий блок коду.Це пришвидшує розробку і оптимізує її адже економить багато ресурсів за рахунок того що не треба запускати всю програму.

- Недоліки Jetpac Compose

  Попри велику кількість переваг Jetpac Compose має деякі недоліки.Більша частина яких пов’язана з тим що це відносно нова технологія.

Одна з проблем у тому що крива навчання вище порівняно з XML. Наступна проблема полягає в тому що через новизну технологію існує не так багато прикладів для роботи з деякими шаблонами і загалом іноді можна зустрітись з проблемою для якої ще немає настільки ефективного рішення як на XML.Ще одною проблема в тому що можуть бути проблеми в роботі з певними старими бібліотеками.

У висновок до всього написаного вище, Jetpac Compose – це революційна технологія,яка значно пришвидшує та полегшує розробку мобільних застосунків.

2.4 Хмарна платформа Firebase

В наш час існує величезна кількість мобільних звстосунків і всі вони стикаються з тим що потребують серверну частину для автентфікації користувачів,синхронізації з пристроями у реальному часі та обробки усіх данних.Оскільки це розповсюджена вимога до застосунків був створений Firebase. Це вже готова інфраструктура створена Google яка надає готове рішення яке дегко інтегрується у мобільних застосунках.

Firebase це хмарна платформа яка допомагає підтримувати, розробляти та маштабувати застосунки.

Деякі з сервісів які він надає:

- зберігання структурованих і неструктурованих даних,
- керування автентифікацією користувачів,
- підтримку роботи в реальному часі (реактивне оновлення даних),
- відправку push-повідомлень,
- аналітику активності користувача,
- логування помилок і продуктивності застосунку,
- обробку зображень, хостинг

Архітектура використання Firebase

- Firebase SDK (Software Development Kit)

Firebase SDK це список бібліотек який дозволяє виконувати такий функціонал як авторизація користувача,читання,запис та оновлення данних в базі данних Firebase,логування помилок та збір аналітики.

- REST API

  REST API використовується тоді коли треба більш широкий функціонал ніж може надати Firebase SDK. Наприклад він використовується у міжсервісній інтеграції,а також у ситуаціях коли по певним причинах потрібно створити запит до Firebase без SDK.

- WebSocket-з’єднання

  Суть WebSocket-з’єднання полягає у тому що данні у користувача оновлюються не тільки після того як користувач надсилає запит до серверу,а й у той момент коли данні оновлюються на сервері.Це відбуваєтья тому що WebSocket підтримує постійне з’єднання з сервером і слідкує за змінами.

2.5 Середовище розробки Android Studio

У сучасній розробців андоїд застосунків використання Android Studio є стандартом.Це зручне середовище розробки створене Google, яке ідеально підходить для створення андроїд-застосунків.

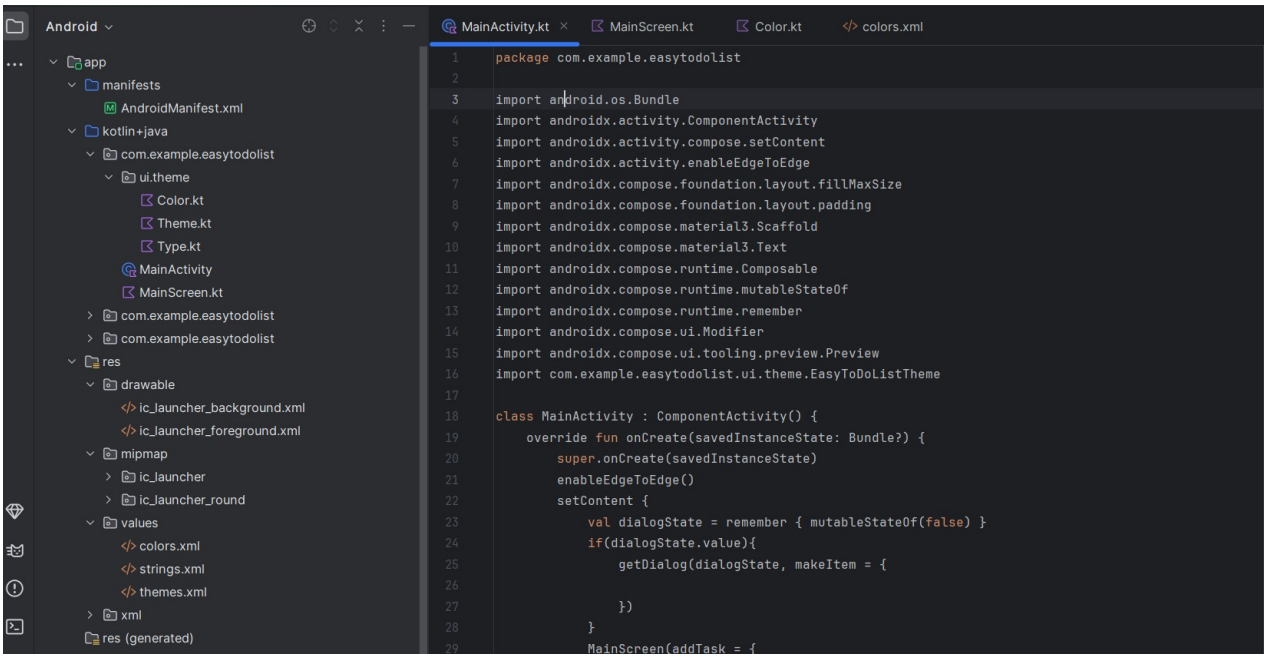

Рисунок1.1 інтерфейс Android Studio

Загальна характеристика Android Studio

Android Studio — це безкоштовне середовище розробки, яке підтримує мови Kotlin, Java та C++, а також новітні інструменти Android Jetpack, зокрема Jetpack Compose. IDE має тісну інтеграцію з усіма компонентами Android SDK, Android Emulator, Gradle, Firebase, а також із системами контролю версій (Git, GitHub).

Основні компоненти Android Studio

- Редактор коду

  Працює з такими мовами як Java та Kotlin.Підтримує навігацію по проекту та підсвітку коду.
- UI Designer та Compose Preview

  Це зручний інструмен який дозволяє переглядати компоненти без запуску усього коду.
- Емулятор Android

  Створює віртуальний пристрій для тестування програми. Підтримує різні версії Android.
- Build-система Gradle

  Це система для зручної збірки усіх потрібних бібліотек.Вона автомтизує процес збірки,тестування та створення .apk файлів.
- Інструменти для аналізу продуктивності

  Memory Profiler, CPU Profiler, Network Profiler.

2.6 Висновки до розділу

В цьому розділі було розказано про основні технології та принципи для розробки адаптивної платформи для організації щоденних записів.
Розгляд переваг та недоліків цих інструментів пояснює причини їх вибору та дозволяє максимально ефективно створювати застосунки такого типу.

Клієнт-хмара – це архітектура де всі дії відбуваються на стороні серверу,а клієнт лише створює запит.Було розглянкуто дворівневу та трирівневу архітектуру.

Jetpac Compose – це сучасний та ефективний інструмент для UI-розробки який дозволяє уникнути дубльованості коду і підвищує читабельність програми.

Firebase – це існуюча реалізація серверної частини,що дозволить не створювати серверну частину самому.

Android Studio – це середа розробки яка має ідеальну сумісність з Jetpac Compose та Firebase,що робить її оптимальною для створення мобільних застосунків.

3. ОПИС РОЗРОБЛЕНОГО АДАПТИВНОГО ЗАСТОСУНКУ ОНЛАЙН-ОРГАНІЗАЦІЇ ЩОДЕННИХ ЗАВДАНЬ В ХМАРНОМУ СЕРЕДОВИЩІ

3.1 Загальна характеристика розробленого застосунку

Розроблений застосунок є повноцінною платформою для розпорядку повсякденними справами.В ньому розроблений фунціонал який дозволяє додавати,видаляти,редагувати та фільтрувати записи.Основна мета цього застосунку у тому щоб створити простий інтуїтивно зрозумілий,але при йцому мінімалістичний застосунок.

Він орієнтований на користувачів які потребують зручний та швидкий інструмент для створення та контролю списку щоденних задач.

Призначення та концепція застосунку

Програма реалізує функціональність персонального цифрового планувальника, якйи надає доступ до списку задач створеного користувачем з будь якої точки світу у будь який час.

Ця програма є гарним інструментом для вирішення таких задач:

- Зберігання завдань в хмарі без прив’язки до конкретного пристрою
- можливість швидкого перегляду та фільтрації завдань;
- отримання повідомлень про прострочені або невиконані завдання;
- наявність єдиного облікового запису користувача.

Інтерфейс розроблений таким чином щоб не первантажувати користувача і бути простим та інтуїтивно зрозумілим.

Фунціональіть:

- Реєстрація та автентифікація користувачів
- Створення редагування та видалення задач
- Зберігання задач у хмарній базі данних
- Синхронізація данних у реальному часі
- Фільтрація завдань

- Нагадування про невиконані задачі
- Адаптивність під різні розміри екранів

Компонентна структура проєкту

Кожна частина програми реалізована у вигляді окремого екрану чи класу.

Перелік усіх компонентів:

MainActivity – відправна точка в програмі,саме вона відповідальна за створення початкового екрану

LoginScreen – реалізує екран для реєстрації,автентифікації та виходу з акаунту користувача

TaskListScreen – реалізує екран де будуть виводитись усі задачі користувача

TaskEditScreen – дозволяє редагувати задачі

Remainder – слідкує за дедлайном задач та створює нагадування при наближенні дедлайну.

Адаптивність інтерфейсу

Дуже важливою вимогою для цієї програми була адаптивність,тобто можливість підлаштовуватись під різний розмір екранів,при необхідності змінювати розмір чи розположення деяких елементів.Вона була реалізована за допомогою Jetpac Compose, а саме таких контейнерів як Column,LazyColumb,Box та інших.

Сценарій використання

Типовий життєвий цикл користувача виглядає так:

1. Користувач запускає застосунок - відкривається LoginScreen.
2. Відбувається автентифікація через Firebase.
3. Після входу відкривається TaskListScreen, де відображаються усі активні завдання.
4. Натискаючи кнопку «Додати задачу», користувач переходить на TaskEditScreen, вводить дані, підтверджує збереження.
5. Задача записується у Firestore та з’являється в загальному списку.
6. . При завершенні задачі користувач змінює її статус — або видаляє повністю.

7. Якщо задача не завершена до вказаної дати — система генерує повідомлення з нагадуванням.

Таким чином розроблена програма відповідає сучасним стандартам інженерії програмного забезпечення.

3.2 Архітектура даних та хмарне сховище Firestore

Застосунок для онлайн організації щоденних завдань базується на хмарному сховищі Firebase,а саме на сервісі FireSrore Cloud – сучасна NoSQL база данних,яка дає можливість синхронізації та обробки данних в реальному часі.

Логічна структура бази даних: колекції та документи

Cloud Firestore використовує ієрархічну модель даних, де вся інформація організована у вигляді вкладених колекцій та документів. Колекція — це набір документів, кожен з яких є окремим об’єктом даних з унікальним ідентифікатором. Документ, своєю чергою, може містити вкладені колекції. Для реалізації функціоналу онлайн-щоденника була побудована наступна структура даних:

bash

users → {uid} → tasks → {taskId}

- users — основна колекція всіх користувачів;
- {uid} — унікальний ідентифікатор користувача, який створюється Firebase Authentication;
- tasks — вкладена колекція, що містить усі завдання конкретного користувача;
- {taskId} — унікальний ідентифікатор окремої задачі.

  Цей підхід забезпечує просту ізоляцію даних користувачів та дозволяє кожному зберігати свої власні записи незалежно від інших. Зчитування задач здійснюється лише в межах поточного авторизованого користувача.

Структура документа задачі

Кожна задача зберігається у вигляді окремого документа у колекції tasks. Кожен елемент має чітку структуру,яка містить основні атрибути,які потрібні для пошуку,збереження та фільтрації задач.

Структура елементу:

- title - коротка назва задачі;
- description – розгорнутий опис або пояснення;
- dueDate – дата та час, до яких завдання повинне бути виконане (тип: Timestamp);
- isDone – мітка яка показує чи виконана задача
- createdAt — час створення завдання.

Взаємозв'язок між користувачем і його задачами

Взаємозв'язок між користувачем і його задачами реалізована таким чином що у кожного користувача є своя персональна колекція tasks,доступ до якої він може отримати тільки після автентифікації за допомогою унікального id користувача.

Це дозволяє реалізувати:

- зберігання персоналізованої інформації;
- повну ізоляцію записів між різними користувачами;
- можливість масштабування без конфлікту ідентифікаторів;
- просту реалізацію логіки доступу.

Firebase Rules: обмеження доступу

У Firestore існує потужний механізм контролю доступу до даних — Firebase Security Rules, який забезпечує перевірку прав на читання та запис на основі ідентифікатора користувача.

Саме він був використаний у програмі для забезпечення безпеки і він гарантує:

- захист персональних даних;
- неможливість доступу одного користувача до чужих задач;
- відповідність вимогам безпеки при розробці багатокористувацьких систем.

Приклади реальних документів у Firestore

На етапі тестування було створено такі приклади завдань:

users/uid_1234abcd/tasks/task_001


```
{
  "title": "Розмова з куратором",
  "description": "Обговорити хід написання дипломної роботи",
  "dueDate": "2025-06-12T10:00:00Z",
  "isDone": false,
    "createdAt": "2025-06-08T09:45:00Z"
}
```


users/uid_5678efgh/tasks/task_012


```
{
  "title": "Купити продукти",
  "description": "Молоко, хліб, овочі",
  "dueDate": "2025-06-08T18:00:00Z",
  "isDone": true,
  "createdAt": "2025-06-07T16:25:00Z"
}
```


Як видно задачі відрізняються,але мають однакову структуру

3.3 Реалізація функціональних модулів

Модуль автентифікації користувача

Це модуль який складається з трьох функцій SignUp – відповідає за реєстрацію користувача,SignIn – відповідає за вхід в існуючий акаунт та SignOut – відповідає за вихід з акаунту.Модуль в якому вони реалізовані називається LoginScreen.

Рисунок 3.1 Код функції signUp

Це реалізація функції signUp яка відповідає за реєстрацію користувача. Вона приймає об’єкт типу FirebaseAuth щоб могти використовувати функції хмарного сервісу Firebase та дві об’єкти типу String які являються імейлом та паролем.Реєстрація відбуваєтья за допомогою однієї з функцій FirebaseAuth а саме createUserWithEmaliAndPassword.

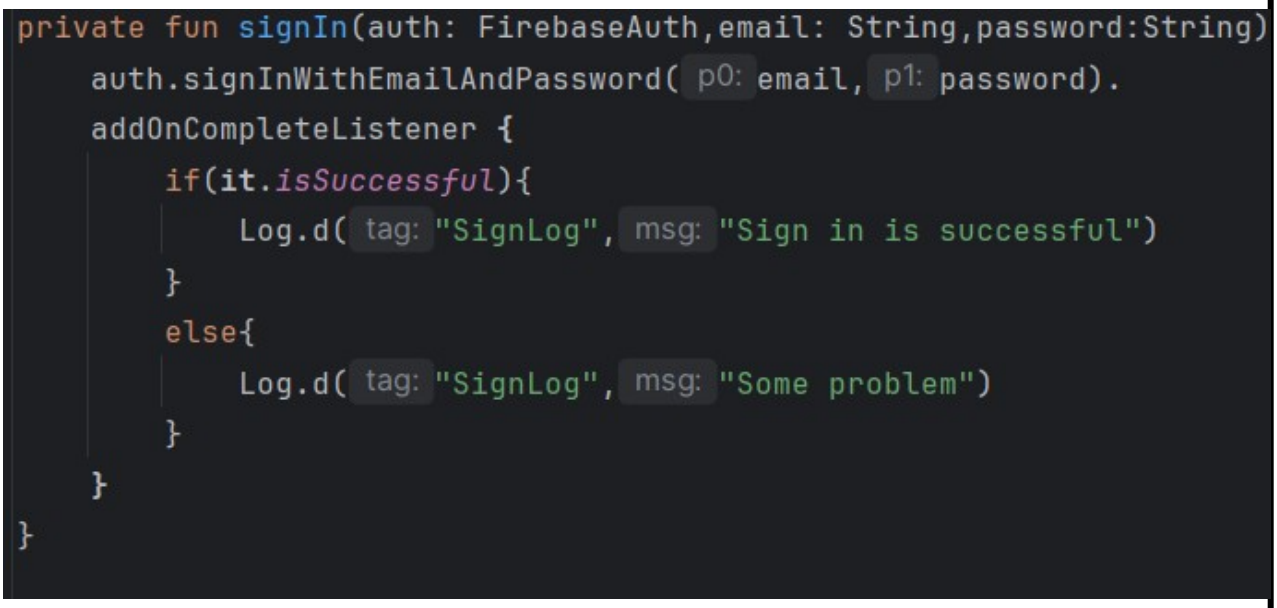

Рисунок 3.2 Код функції signIn

Це реалізація функції SignIn яка відповідає за вхід в акаунт.

Для реалізації входу в акаунт використовується одна з функцій FirebaseAuth, а саме signInWithEmailAndPassword

```
private fun signOut(auth: FirebaseAuth){
    auth.signOut()
}
```

Рисунок 3.2 Код функції signOut

Реалізація функції signOut, яка відповідає за вихід з акаунту.

Оскільки ця функція не потребує особистих данних то на вхід поступає тільки об’єкт auth типу FirebaseAuth.

Модуль керування задачами

Це модуль який відповідає за створення,редагування.та фільтрацію задач

Створення задачі

При натисканні на кнопку «Add task» відкривається форма введення, яка дозволяє користувачеві ввести:

- назву завдання (title);
- опис (description);
- дату дедлайну (dueDate);
- статус (isDone);

Після заповнення форми і натискання кнопки «Save», відбувається валідація введених даних та надсилання інформації у Firestore:

```
data class Task(
    val title: String = "",
    val description: String = "",
    val dueDate: Long = System.currentTimeMillis(),
    val isDone: Boolean = false,
    val createdAt: Long = System.currentTimeMillis()
)
```

Рисунок 3.4 Data class Task

Це код дата класу Task,саме за його структурою створюється завдання.

```
fun createTask(task: Task) {
    val currentUser = auth.currentUser ?: return

    val taskMap = hashMapOf(
        "title" to task.title,
        "description" to task.description,
        "dueDate" to task.dueDate,
        "isDone" to task.isDone,


    )
```

Рисунок 3.5 код функції createTask

Ця функція створює завдання по шаблону класу Task

```
db.collection( collectionPath: "users")
    .document( documentPath: currentUser.uid)
    .collection( collectionPath: "tasks")
    .add( data: taskMap)
    .addOnSuccessListener {
        creationState.value = "Task created"
    }
    .addOnFailureListener {
        creationState.value = "Error"
    }
```

Рисунок 3.6 код відповідальний за відправку данних на сервер

Цей фрагмент коду відповідає за створення нового докумнту на на сервері Firebase.

Цей процес відбувається в такому порядку:

- db.collection("users") – звертаємось до колекції users де містяться всі данні всіх користувачів.
- .document(currentUser.uid) – обираємо документ який належить саме цьому користувачу,за допомогою унікального id
- .collection("tasks") – звертаємось до колекції tasks,кожна задача в цій колекції це окремий документ

- .add(taskMap) – додає новий документ до колекції tasks
- .addOnSuccessListener {

creationState.value = "Завдання створено"

} – це код який виконується у разі успішного виконнаня і повідомляє про успішне виконання користувача.

Модуль редагування та видалення

Користувач може відкрити цей екран ввести в поле оновдені данні,а потім натиснути кнопку “Update” і данні оновляться.

```
db.collection( collectionPath: "users").document( documentPath: userId)
    .collection( collectionPath: "tasks").document( documentPath: taskId
    .update(updatedFields)
```

Рисунок 3.7 Код для оновлення інформації у базі данних

Цей фрагмент коду відповідає за оновлення данних він працює за таким принципом:

Звертаємось до колекції users

Обираємо конкретного користувача за унікальним id

Звертаємось до колекції tasks

Обираємо конкретне завдання за допомогою унікального id

Оновлюємо данні викликавши функцію update.

```
db.collection( collectionPath: "users").document( documentPath: userId)
    .collection( collectionPath: "tasks").document( documentPath: taskId)
    .delete()
```

Рисунок 3.8 Код для видалення інформації з бази данних

Цей фрагмент коду відповідає за видалення запису.Працює за таким самим принципом як і код для оновлення,але в кінці замість функції для оновлення викликає функцію delete.

Модуль фільтрації

Це модуль який відповідає за фільтрацію задач залежно від того виконані вони чи ні а також фільтрації по часу.

```
db.collection( collectionPath: "users").document( documentPath: userId)
    .collection( collectionPath: "tasks")
    .whereEqualTo( field: "isDone",  value: false)
    .orderBy( field: "dueDate")
    .get()
```

Рисунок 3.9 Код який відповідає за фільтрацію

Цей фрагмент коду проводить фільтрацію задач

Він працює за таким принципом:

- Звертається до колекціїї users
- Знаходить конкретного користувача за унікальним id
- Звертається до колекції tasks
- Обирає задачі які ще не виконані
- Сортує задачі за датою
- Повертає список задач

Модуль нагадувань

Цей модуль реалізований за допомогою WorkManager – механізму фонових задач в Android Studio.Він перевіряє усі задачі термін яких уже пройшов і надсилає нагадування про ті які ще не виконані.Перевіряє раз у день в зазначений час.

Реалізація функціональних модулів у мобільному застосунку забезпечує повний набір можливостей для ефективної організації щоденних завдань. Модульний підхід, використання Firebase та архітектури MVVM дозволили створити стабільну, гнучку й масштабовану систему, що легко адаптується до потреб користувачів. Ретельна організація кожного процесу — від реєстрації до нагадування – сприяє інтуїтивному користуванню та високій якості кінцевого продукту.

3.4 Користувацький інтерфейс

У сучасних мобільних додатках інтерфейс користувача (UI) є не лише засобом взаємодії, а й важливою складовою загального користувацького досвіду (UX). Він визначає не лише зовнішній вигляд, а й інтуїтивність, доступність і швидкість виконання дій. У процесі розробки адаптивного мобільного застосунку «Онлайн-організація щоденних завдань» особливу увагу було приділено створенню простого, естетично привабливого та зручного UI.

Опис основних екранів застосунку

Екран авторизації (LoginScreen)

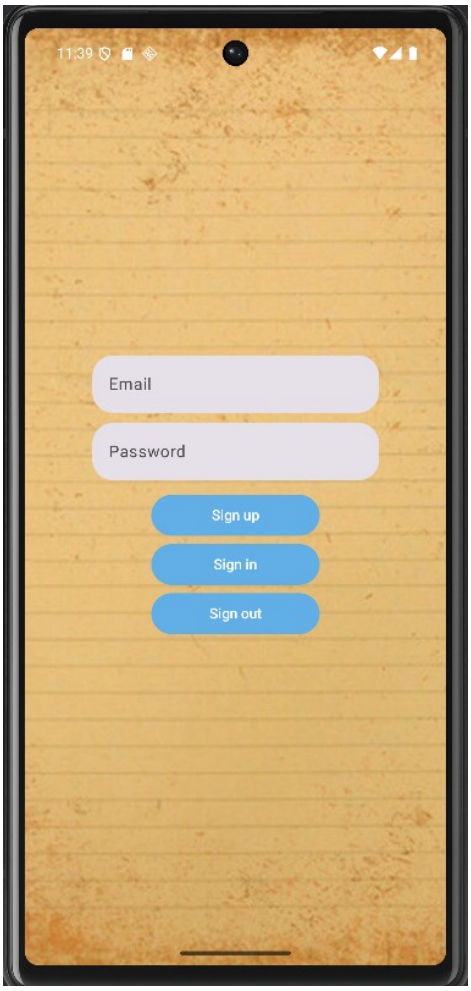

Рисунок 3.10 Інтерфейс екрану LoginScreen

Після запуску застосунку з’являється цей екран який дозволяє:

- зареєструватись,
- авторизуватись
- вийти з акаунту.

Екран з задачами (TaskListScreen)

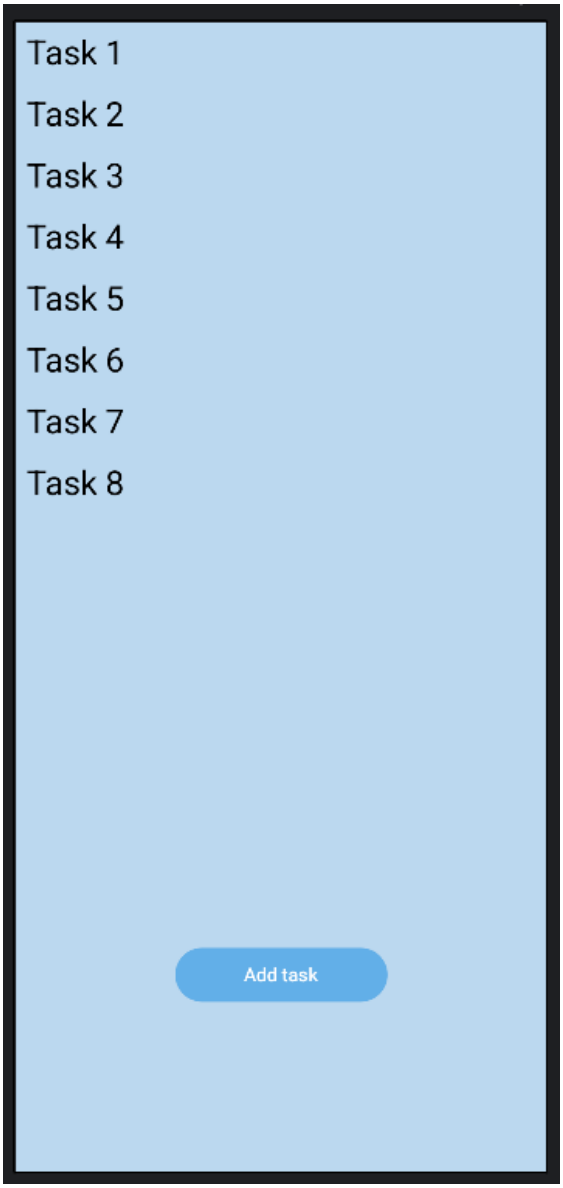

Рисунок 3.11 TaskListScreen

При успішній авторизації користувач потрапляє на екран з списком усіх задач.Екарн включає в себе:

- Верхню частину де знаходиться список усіх задач
- Кнопку Add task

Завдяки використанню Jetpack Compose, список оновлюється автоматично при кожній зміні даних у базі.

Екран для створення та редагування задач(TaskEditScreen)

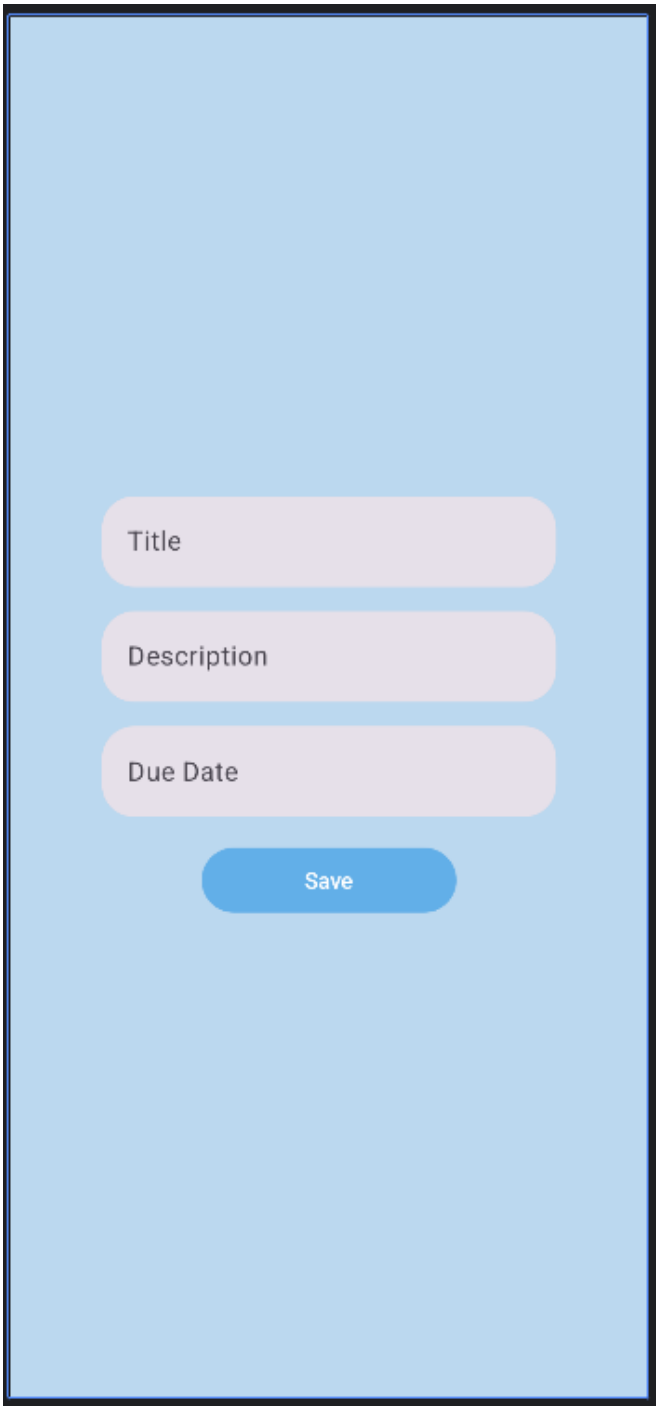

Рисунок 3.12 TaskEditScreen

Призначення цього екрану створення та редагування задач.Для створення задачі необхідно заповнити поля:

- Title
- Description
- DueDate

І після заповнення цих полів натиснути кнопку “Save”

3.5 Налаштування Firebase в застосунку

Створення проєкту в консолі Firebase

Перш за все був створений проект в Firebase з назвою “DailyTasks” Створити проект необхідно щоб мати можливістьпрацювати з цим сервісом у Android Studio.

Реєстрація застосунку в Android Studio

Наступним кроком пілся створення проєкту була його реєстрація в Android Studio потім завантаження файлу google-services.json і додавання його в корінь проєкту.

Зміни у файлах gradle

```
[libraries]
firebase-bom = { group = "com.google.firebase", name = "firebase-bom", version.ref = "bomVersion" }
firebase-firestore = { group = "com.google.firebase", name = "firebase-firestore-ktx"}
firebase-auth = { module = "com.google.firebase:firebase-auth"}
```

Рисунок 3.13 Зміни в файлі libs.versions.toml

Список доданих бібліотек та їх призначення:

Firebase-bom – це бібліотека яка слідкує і автоматично оновлює версії інших бібліотек пов’язаних з Firebase

Firebase-firestore – це бібліотека яка необхідна для створення бази данних де будуть зберігатись данні про користувачів та їх задачі

Firebase-auth – це бібліотека як потрібна для реєстрації та автентифікації користувачів.

```
implementation(platform( dependencyProvider: libs.firebase.bom))
implementation(libs.firebase.firestore)
implementation(libs.firebase.auth)
```

Рисунок 3.14 Зміни в файлі build.gradle(app module)

Налаштування Firebase Firestore

У консолі Firebase:

- Вибрано Cloud Firestore
- Натиснуто Create database
- Обрано режим Start in test mode (тільки під час розробки)
- Обрано локацію зберігання (наприклад, europe-west1)

Firebase Rules

Firebase rules – це система правил безпеки яка вказує хто може записувати,читати та змінювати данні.

Для застосунку були написані такі правила:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId}/tasks/{taskId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

Рисунок 3.15 Firebase rules

3.6 Схема навігації та взаємодії між екранами

Процес взаємодії користувача з додатком організовано у вигляді послідовного потоку, який відбувається в кілька етапів:

Стартова точка — екран автентифікації (LoginScreen)

- Користувач вводить email і пароль.
- У разі успішного входу — переходить до списку задач.
- У разі помилки — відображається відповідне повідомлення.

Головний екран — список завдань (TaskListScreen)

- Відображаються задачі користувача.
- Доступна можливість додати нове завдання або редагувати наявне.
- Можна застосувати фільтри, відзначити задачу як виконану.

Екран створення задачі (TaskEditScreen)

- Користувач вводить данні нової задачі
- Натискає кнопку “Save”
- Після натиску на кнопку “Save” задача додається до списку і відбувається перехід на екран TaskListScreen

3.7 Потенціал для розширення застосунку

Застосунок розроблений для дипломного проєкте реалізує базовий функціонал адаптивного застосунку для організації щоденних завдань,він розроблений такиим чином що може бути легко розширеним і можнливо легко додаюти нові можливості без потреби суттєвих змін у кодові. Нижче наведено кілька перспективних напрямків розширення, кожен із яких покликаний підвищити зручність використання, функціональну повноту й конкурентоспроможність затосунку.

Додавання тегів,категорій,пріорітетності

Основна задача застосунку це управління щоденними задачами і контролювати та організовувати задачі набагато легше коли всі задачі можна сортувати і робивати за різними категоріями.
окрему колекцію categories, що дозволить динамічно зберігати користувацькі категорії з відповідними кольорами та піктограмами.

- Категорії задач

Категорії – це загальні групи які створені за певною спільною рисою. Вони сприяють кращій орієнтації в застосунку.Прикладами категорій є робота,навчання,подорожі,спорт або будб які інші які обере користувач. Запровадження категорій дозволяє реалізувати:

- групування задач за напрямками життя чи проектами;
- фільтрацію задач в інтерфейсі за конкретною категорією;
- статистичний аналіз навантаження користувача в тій чи іншій сфері

(наприклад, % задач пов’язаних з роботою).

З технічної точки зору,це можна реалізувати створивши ще одне поле в дата класі Tasks наприклад назвавши його category.

- Теги (мітки)

  Теги – це більш гнучка форма класифікації коли користувач додає позначки використовуючі певні ключові слова.

  Відмінність тегів від категорій полягає в тому, що:

- задачі можуть мати декілька тегів одночасно;
- теги зазвичай є коротшими, мають форму хештегу й використовуються для швидкої ідентифікації суті задачі.

На рівні бази даних теги можуть зберігатися як масив рядків

- Рівень пріоритету

У кожної задачі можна зробити рівень пріоритету щоб було легше визначитись в якому порядку вирішувати задачі

Виділяють такі стандартні рівні пріоритету:

- Високий – критично важливі завдання (горять дедлайни, обов’язкові для виконання);
- Середній – важливі, але не термінові задачі;
- Низький – задачі які не потребують термінового втручання

Для реалізацій можна надати кожному пріорітиту свій колір і користуючись можливостями Jetpac Compose змінвати колір фону щоб одразу було візуально видно пріоритет задачі.

Підтримка push-сповіщень

Ще одним варіантом розвитку є створення сповіщень які періодично приходять користувачу незалежно від того ввімкниний застосунок чи ні.

Функціональні можливості push-сповіщень

- Нагадування про наближення дедлайну задачі

Можна зробити так щоб вспливали повідомлення за пару годин до дедлайну для нагадування про задачу

- Сповіщення про нові задачі у спільних списках

У перспективі можна зробити можливість створювати спільні списки для виконання групових задач і відповідно надсилати сповіщення кожен раз коли хтось додає або виконує задачу з списку.

- Оповіщення про оновлення функціоналу застосунку

При зміні функціоналу або інтерфейсу надсилається повідомлення про внесені зміни

Реалізувати можна за допомогою Firebase Cloud Messaging – це технологія яка дозволяє надсилати повідомлення навть коли застосунок вимкнений.

Голосове введення завдань

Голосове введення зараз дуже розповсюджене,в багатьох сучасних месенджерах можна відправляти голосові повідомлення.Можливість голосового вводу дозволить вводити задачі в будь який момент часу.

Переваги голосового створення задач:

- користувач може створити нове завдання за кілька секунд, не вводячи текст вручну;
- значно зменшується час створення задачі, що підвищує продуктивність;
- розширюється доступність застосунку для людей з порушеннями моторики;
- можливе використання в начебто незручних ситуаціях наприклад за кермом, у русі, під час занять.

Можна реалізувати за допомогою SpeechRecognizer API

Синхронізація з Google Calendar

Google Calendar – один з найпопулярніших сервісів для плануваняя,тому інтеграції з ним це логічне і перспективне доповнення до будь якої платформи для організації щоденних задач.

Перваги від інтеграції:

- Імпорт подій з Google Calendar у застосунок

  Користувач надає дозвіл через OAuth2, після чого в нього з’являється можливість додавати в застосунок події з календаря.

  - Експорт задач у вигляді подій у Google Calendar

    Кожна задача може автоматично додаватись до календаря з почтаковим часом,тегами,та описом.Завдяки цьому всі активності користувача будуть в одному місці.

  - Синхронізація змін у двох напрямках

    Зміна дати,тегу чи опису в календарі буде змінюватиту саму задачу в застосунку і навпаки.

  - Відображення задач у форматі календаря

    Список задач буде у форматі календаря,а не списку,що дозволить планувати не тільки щоденні задачі,а й задачі з перспективою на декілька днів чи тижнів.

Підтримка спільного доступу до завдань

Задумкою цього застосунку був персональний організатор щоденних завдань.Але у виді розширення функціоналу можна розглянути спільний доступ до завдань.Це розширить можливості і дасть змогу працювати над складнішими задачами.

Основний функціонал спільного списку:

- переглядати задачі — отримувати оновлення в реальному часі, спостерігати за прогресом команди;
- створювати або редагувати їх — додавати нові задачі, вносити правки до заголовку, опису, дедлайну;
- змінювати статус виконання — наприклад, "нове", "в процесі", "виконано";
- залишати коментарі — вести короткі обговорення під кожною задачею, надаючи додатковий контекст;

- отримувати сповіщення про зміну стану списку — наприклад, при додаванні нової задачі, оновленні дедлайну або зміні статусу.

3.8 Висновки

У результаті виконання дипломного проєкту був розролений адаптивний мобільний застосунок для онлайн-організації щоденних завдань, що функціонує у хмарному середовищі з використанням платформи Firebase. Реалізований застосунок відповідає всім сучасним вимогам.Він інтерактивний,адаптивний до різних розмірів екранів та підтримує синхронізацію в реальному часі.

На першому етапі проєкту було проведено аналіз актуальності теми,та визначено потребу користувачів у адаптивному додатку для організації щоденних справ з такими функціями як редагування,фільтрація,а також нагадування про закінчення дедлайнів.Були досліджені різні види архітектури та сервіси для взаємодії з базами данних та був обраний Firebase.

Було створено адаптивний інтерфейс за допомогою Jetpack Compose, що забезпечує гнучкість UI-компонентів, включаючи підтримку Material Design. Ці технології забезпечують адптивність екранів,навігацію між екранами та інтерактивні списки.

На рівні архітектури реалізовано підхід MVVM (Model-View-ViewModel), що забезпечив чітке розділення логіки, полегшив тестування, масштабування та модифікацію коду. База даних проєктована у вигляді колекцій та документів у Firestore.

Також було розглянуто потенціал масштабування застосунку: можливість додавання тегів, категорій, системи пріоритетів, push-сповіщень через Firebase Cloud Messaging, голосового введення задач, інтеграції з Google Calendar та розробки вебверсії. Це дозволяє проєкту у майбутньому перерости в повноцінну платформу персонального та командного керування часом.

Застосунок протестовано на актуальних версіях Android, верифіковано коректну роботу механізмів реєстрації, збереження, пошуку та нагадування. Результати підтвердили відповідність функціоналу початковим вимогам.

Таким чином, поставлену у дипломному проєкті мету – створити адаптивний мобільний застосунок для організації щоденних завдань з хмарним зберіганням і підтримкою базової логіки управління задачами – було успішно виконано. Розроблений застосунок має чітку архітектуру, реалізовану функціональність, адаптивний інтерфейс, а також закладений фундамент для подальшого розвитку, масштабування та практичного застосування у реальних умовах.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1.Neil Smyth. Android Studio 4.2 Development Essentials - Kotlin Edition. PayloadMedia, 2021. — 801 p.

2. Google Developers. Jetpack Compose – Modern toolkit for building native UI [Електронний ресурс]. – Режим доступу: https://developer.android.com/jetpack/compose.

3. Wargo M. Learning Android Application Development. Addison-Wesley Professional, 2015. — 456 p.

4. Google Firebase. Firebase Documentation [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs.

5. Google Developers. Guide to App Architecture (MVVM) [Електронний ресурс]. – Режим доступу: https://developer.android.com/topic/architecture.

6. Antonio Leiva. Kotlin for Android Developers. Leanpub, 2017. — 261 p.

7. Accompanist — Jetpack Compose libraries [Електронний ресурс]. – Режим доступу: https://google.github.io/accompanist/.

8. Navigation Compose — Android Jetpack [Електронний ресурс]. – Режим доступу: https://developer.android.com/jetpack/compose/navigation.